**SQL Cheat Sheet**

**Course 1: SQL for Data Science**

|  |  |  |
| --- | --- | --- |
| **Name** | **Code** | **Explanation & Links** |
| SELECT  FROM | SELECT  prod\_name  ,prod\_id  ,prod\_price  FROM Products;  SELECT \*  FROM Products  LIMIT 5; | Columns to be returned (REQUIRED)  Table from which to retrieve data  Selecting all columns  But limiting to 5 |
| CREATE TABLE | CREATE TABLE Shoes  (  Id char(10) PRIMARY KEY,  Brand char(10) NOT NULL,  Type char(250) NOT NULL,  Price decimal(8,2) NOT NULL,  Desc Varchar(750) NULL  ); | Creating Table  NULL -> absence of anything  Empty -> empty string  Null values != Empty strings  Primary keys != NULL |
| Temporary Table | CREATE TEMPORARY TABLE Sandals AS  (  SELECT \*  FROM shoes  WHERE shoe\_type = ‘sandals’  ) |  |
| Comment | -- this is single line comment  /\*  This is  Multiple line comment  \*/ |  |
| Filter | SELECT  column\_name  ,column\_name  FROM table\_name  WHERE column\_name operator value; | Increase query performance  Operator: =, <, >, <> not equal, >=, <=, BETWEEN, IS NULL  Row-level filtering |
| IN | SELECT  ProductID  ,UnitPrice  ,SupplierID  FROM Products  WHERE SupplierID IN (9,10,11); | Specifies a range of conditions  Comma delimited list of values  Enclosed in () |
| OR | SELECT  ProductID  ,UnitPrice  ,SupplierID  FROM Products  WHERE  (SupplierID = 9 OR SupplierID = 11)  AND UnitPrice > 15; | OR is performed before AND |
| NOT | SELECT \*  FROM Employees  WHERE  NOT City = ‘London’  AND NOT City = ‘Seattle’; |  |
| Wildcards | WHERE size LIKE ‘%pizza’  Output:  spizza  mpizza | ‘%Pizza’ anything ending with pizza  ‘Pizza%’ anything after pizza  ‘%Pizza%’ anything before and after pizza  ‘S%E’ anything starting with “S” and ending with “E”  ‘t%@gmail.com’ any gmail starting with t  \_ sometimes can be used instead of %, also []  Wildcards take longer to run |
| ORDER BY | SELECT something  FROM database  ORDER BY characteristic  ORDER BY 2,3  ORDER BY column\_name ASC | Output sort order  Sorting by column position |
| Math | SELECT  ProductID  ,UnitsOnOrder  ,UnitPrice  ,UnitsOnOrder \* UnitPrice  AS Total\_order\_Cost  FROM Products; |  |
| Aggregate | SELECT AVG(UnitPrice) AS avg\_price  FROM products;  SELECT SUM(UnitPrice) AS total\_prod\_price  FROM Products;  SELECT COUNT(DISTINCT CustomerID)  FROM Customers | Rows containing with Null values are ignored by AVG function  COUNT (\*) - counts all the rows in a table containing value/NULL  COUNT(column) - counts all the rows in a specific column containing values and ignoring NULL  MAX(UnitPrice), MIN(UnitPrice)  SUM(UnitPrice)  \*MAX(string) will give the one that is not NULL among two strings  Cannot use DISTINCT on COUNT(\*) |
| GROUP BY | SELECT  Region  ,COUNT(CustomerID) AS total\_customers  FROM Customers  GROUP BY Regions;  SELECT  CustomerID  ,COUNT(\*) AS orders  FROM Orders  GROUP BY CustomerID  HAVING COUNT(\*) >= 2; | Output sort order  GROUP BY can contain multiple columns  NULLS are grouped together in GROUP BY  WHERE filters before data is grouped  HAVING filters after data is grouped  Rows eliminated by WHERE will not be included in the group  GROUP BY does not work without an aggeragate function |
| LAG | SELECT  LAG(car\_rentals)  OVER (ORDER BY date\_id)  AS previousDayRental  SELECT  LEAD(car\_rentals)  OVER (ORDER BY date\_id)  AS previousDayRental  LAG(expression, offset, default)  OVER (PARTITION BY partition\_expression  ORDER BY sort\_expression) | LAG - Compare with previous row value |
| Subqueries | SELECT  CustomerID  ,CompanyName  ,Region  FROM Customers  WHERE customerID IN  (SELECT customerID  FROM Orders  WHERE Freight > 100 );  SELECT  id  FROM (  SELECT  id,  temperature  ,LAG(temperature) OVER (ORDER BY recordDate) AS prevTemp  FROM Weather  ) AS Subquery  WHERE temperature > prevTemp | Queries embedded into other queries  Merge data from multiple tables  Additional filtering criteria  Too many queries make performance slower  Only retrieve a single column at a time  Code indenting site: [www.poorsql.com](http://www.poorsql.com) |
| Cartesian (Cross) Join | SELECT  product\_name  ,unit\_price  ,company\_name  FROM suppliers CROSS JOIN products; | Output: #Joins in Table 1 \* #Rows in Table 2  Not frequently used  Computationally taxing |
| Aliases | SELECT  vendor\_name  ,product\_name  ,product\_price  FROM Vendors AS v, Products AS p  WHERE v.vendor\_id = p.vendor\_id | SQL aliases give a table or a column a temporary name => more readable |
| Inner Join | SELECT  suppliers.CompanyName  ,ProductName  ,UnitPrice  FROM Suppliers  INNER JOIN Products  ON Suppliers.supplierid = Products.supplierid  SELECT  o.OrderID  ,c.CompanyName  ,e.LastName  FROM  ( (Orders o INNER JOIN  Customers c  ON o.CustomerID = c.CustomerID)  INNER JOIN Employees e  ON  o.EmployeeID = e.EmployeeID ); | Set AB |
| Self Join | SELECT column\_name(s)  FROM  Table1 T1, table2 T2  WHERE condition;  SELECT  A.CustomerName AS CustomerName1  ,B.CustomerName AS CustomerName2  ,A.City  FROM  Customers A  ,Customers B  WHERE A.CustomerID = B.CustomerID  AND A.City = B.City  ORDER BY A.City; | Take one table and treat it like two separate tables to  Join the original table to itself  Example application: match customers from the same city |
| Left Join | SELECT  C.CustomerName  ,O.orderID  FROM Customers C  LEFT JOIN Orders O  ON C.CustomerID = O.CustomerID  ORDER BY C.CustomerName; | SQL Lite only does Left Join  Result is NULL from the right side when there is no match  Set A |
| Right Join | SELECT  Orders.OrderID  ,Employees.LastName  ,Employyees.FirstName  FROM Orders  RIGHT JOIN Employees  ON Orders.EmployeeID = Employees.EmployeeID  ORDER BY Orders.OrderID | Result is NULL from the left side when there is no match  Set B |
| (Full) Outer Join | SELECT  Customers.CustomerName  ,Orders.OrderID  FROM Customers  FULL OUTER JOIN Orders  ON Customers.CustomerID = Orders.CustomerID  ORDER BY Customers.CustomerName | Set AUB |
| Unions | SELECT column\_name(s)  FROM table1  UNION  SELECT column\_name(s) FROM table2;  SELECT City, Country FROM Customers  WHERE Country = ‘Germany’  UNION  SELECT City, Country FROM Suppliers  WHERE Country = ‘Germany’  ORDER BY City; | Combine the result-set of two or more SELECT statements  Each SELECT statement within UNION must have the same #columns, similar data types and same order |
|  | | |
| String |  | Retrieve the data in the format needed  Client vs server formatting  Support joins |
| String character length | SELECT LENGTH(my\_string) AS character\_count  FROM table |  |
| Concatenate | SELECT  CompanyName,  ContactName,  CompanyName || ‘ (‘|| ContactName||’)’  FROM customers |  |
| Trim | SELECT TRIM(“ You the best. ”) AS TrimmedString; | Trim the leading or trailing space from a string: TRIM, RTRIM, LTRIM |
| Substring | SUBSTR(  string name,  string position,  number of characters to be returned  )  SELECT  First\_name,  SUBSTR(First\_name,2,3)  FROM employees  WHERE department\_id = 60; |  |
| Upper and Lower | SELECT UPPER(column\_name)  FROM table\_name; | UPPER(col\_name)  LOWER(col\_name)  UCASE(col\_name) |
| Date and Time string | DATE(timestring, modifier, modifier, …)  TIME(timestring, modifier, modifier, …)  DATETIME(timestring, modifier, modifier, …)  JULIANDAY(timestring, modifier, modifier, …)  STRFTIME(format, timestring, modifier, modifier,…) | Time string possible formats:  YYYY-MM-DD  YYYY-MM-DD HH:MM  YYYY-MM-DD HH:MM:SS  YYYY-MM-DD HH:MM:SS.SSS  YYYY-MM-DDTHH:MM  YYYY-MM-DDTHH:MM:SS.sss  HH:MM  HH:MM:SS  HH:MM:SS.SSS  Modifiers:  NNN days start of year  NNN hours start of day  NNN minutes weekday B  NNN.NNNN seconds unixepoch  NNN months localtime  NNN years utc  Start of month |
| String format time function STRFTIME | SELECT Birthdate  ,STRFTIME(‘$Y’, Birthdate) AS Year  ,STRFTIME(‘$Y’, Birthdate) AS Month  ,STRFTIME(‘$Y’, Birthdate) AS Day  FROM employees |  |
| Date | SELECT DATE(‘now’)  SELECT STRFTIME(‘%Y %m %d’, ‘now’)  Select Birthdate  ,STRFTIME(‘$Y’, Birthdate) AS Year  ,STRFTIME(‘$Y’, Birthdate) AS Month  ,STRFTIME(‘$Y’, Birthdate) AS Day  ,(DATE(‘now’) - Birthdate) AS Age  FROM employees | Current date  Calculate age from now |
| DATE\_SUB | DATE\_SUB(date1, INTERVAL 1 DAY) | Get the previous date |
| Case Statement | CASE WHEN C1 THEN E1  WHEN C2 THEN E2  . . .  ELSE [result else]  END  SELECT  Employeeeid  ,firstname  ,lastname  ,city  ,CASE City  WHEN ‘Calgary’ THEN ‘Calgary’  ELSE ‘Other’  END Calgary  FROM Employees  ORDER BY LastName, FirstName; |  |
| Case Statement - Search | SELECT trackid, name, bytes,  CASE  WHEN bytes < 300000 THEN ‘small’  WHEN bytes >= 300000 AND  Bytes <= 500000 THEN ‘medium’  ELSE ‘Other’  END bytescategory  FROM tracks;  SELECT  (CASE WHEN parameter\_name = ‘item\_id’  THEN CAST(parameter\_value AS INT)  ELSE NULL  END) AS item\_id |  |
| Views | CREATE [TEMP] VIEW [IF NOT EXISTS] view\_name(column-name-list)  AS select-statement;  CREATE VIEW my\_view  AS  SELECT  r.regiondescription  ,t.territorydescription  ,e.Lastname  ,e.Firstname  ,e.Reportsto  FROM Region r  INNER JOIN Territories t on r.regionid = t.regionid  INNER JOIN Employees e on e.employeeid = e.EmployeeID  SELECT \*  FROM my\_view  DROP VIEW my\_view;  SELECT  count(territorydescription)  ,Lastname  ,Firstname  FROM my\_view  GROUP BY Lastname, Firstname; | A stored query  Can add or remove cols without changing schema  Use it to encapsulate queries  View will be removed after database connection has ended |
| Data Governance and Profiling | */\**  *Governance Best Practices: (Non-code)*  *Understand your read and write capabilities*  *Clean up your environments*  *Understand your promotion process \*/* | *Column Data Profile: (Non-code)*  *Col data type*  *#distinct values*  *#rows with NULL values*  *Descriptive statistics: max, avg, std dev for col* |
| SQL for Data Science | \*/  Data Understanding:  Most important step  Understanding relationships in data  NULL values  String values  Dates and times  \*/ | Business / Subject Area Understanding:  Understanding where data joins are  Differentiating integers from strings  Investing time to understand the business |

**Course 2: Data Wrangling, Analysis and AB Testing**

|  |  |  |
| --- | --- | --- |
| get\_json | SELECT  get\_json(raw.viewed\_user\_id) AS viewed\_user | if data is formatted in JSON  Use JSON extraction function a/c to SQL |
| GROUP By Check | SELECT  categorical\_column,  COUNT(\*)  FROM  example\_table  GROUP BY  categorical\_column | Use to troubleshoot  Check a table or column is not empty  Helpful for all types of categories |
| NULL Check | SELECT  \*  FROM  example\_table et  JOIN  other\_example\_table oet  ON  oet.user\_id = et.user\_id  **AND**  **et.user\_id IS NOT NULL** | To avoid problem when  NULL values on left table = NULL values on right table => gets joined up |
| COALESCE | SELECT  COALESCE(parent\_user\_id, user\_id) AS original\_user\_id,  parent\_user\_id,  user\_id  FROM  dsv1069.users | Allows to replace in case that it is NULL  \*available in some versions of SQL  use user\_id when parent\_user\_id is NULL |
| Data Types | 1. Numbers - flexible  2. Text - categorization  3. Dates and Times - comparison | |
| Automated Query | INSERT INTO user\_growth  SELECT  ‘{{ds}}’ AS day,  new.new\_added\_users,  ...  ...  ...  FROM  (SELECT  ...  WHERE  date(created\_at) = ‘{{ds}}’  ) new |  |
| New Terms | Dependency: When the data in query refers to data in a preceding table  Stale: When data in a table doesn’t reflect the most up-to-date information  ETL: extract transform load  Job: The task given to a database to perform ETL  Pipeline: Several Table Creation/Update tasks in a specific order  **Backfill**: To run a table creation/update task on a range of dates in the past | |
| Hierarchy of Data |  | |
| Create User Info Table | Snapshot Table New Columns  created\_today  Is\_deleted  deleted\_today  has\_ever\_ordered  ordered\_today  date | Upsteam Dependencies:  Users table  Orders table  Downstream Dependencies:  Dashboard tables  Prediction scores computed using these features |
| Liquid Tags - Variable column | {% assign ds = ‘2018-01-01’ %}  SELECT  id,  ‘{{ds}}’ AS variable\_column  FROM  users |  |
| Data Engineering Questions | 1. How do you know the table contains the data you expect?  - GROUP BY day  - ?  2. How do insert by day?  -  3. How to backfill the data? | |
| Terminology - Partitions | Hadoop is a whole ecosystem of tools that work with a distributed file system (HDFS)  Hive takes a SQL-like code (HQL) and turns it into a map-reduce job that can run on a distributed file system.  Partitions make  - updates faster  - retrieval faster  - joins faster  When to partition? Example.  WHERE day >= data\_add(CURDATE(), INTERVAL -1 month)  -- this will help avoid a full table scan | |
| How to write Big Queries  Sacffolding | Step 1: Figure Out Potential Questions  Step 2: Identify the Tables You Need to Build It  Step 3: Build Subqueries  Step 4: Test Joins, Then Add Columns  Step 5: Clean Up Formatting  Step 6: Give Columns Distinct Names | |
| Windowing Functions | SELECT  user\_id,  invoice\_id,  paid\_at  FROM  dsv1069.orders  GROUP BY  user\_id -- this does not work  ORDER BY  paid\_at  SELECT \*  FROM  (  SELECT  user\_id,  invoice\_id,  paid\_at,  DENSE\_RANK() OVER (PARTITION BY user\_id ORDER BY paid\_at ASC) AS order\_num  FROM  dsv1069.orders  ) sub  WHERE order\_num = 1 -- this has to be outside of the query | GROUP BY does not work without an aggerage function  RANK() OVER (PARTITION BY \_\_ ORDER BY \_\_) as rank  ROW\_NUMBER() OVER(PARTITION BY \_\_ ORDER BY \_\_) as row\_number  SUM() OVER(PARTITION BY \_\_ ORDER BY \_\_) as running\_total  COUNT() OVER(PARTITION BY \_\_ ORDER BY \_\_) as running\_count |
| AB Testing | Branch of statistics: hypothesis testing  In ML, AB testing is a final step in the algorithm development process  AB testing ia a SQL project that moves beyond a single query. | |

**Course 3: Distributed Computing with Spark SQL**

|  |  |  |
| --- | --- | --- |
| mount data | %run ../Includes/Classroom-Setup |  |
| CREATE A DATABASE FOR CREATING TABLE | CREATE DATABASE IF NOT EXISTS Databricks  USE Databricks |  |
| CREATE TABLE | CREATE TABLE IF NOT EXISTS fireIncidents  USING csv  OPTIONS (    header "true",    path "/mnt/davis/fire-incidents/fire-incidents-2016.csv",    inferSchema "true"  ) |  |
| USE `` for column name with space | -- TODO  SELECT `Incident Number`  FROM fireIncidents  WHERE `Incident Date` = DATE('2016-04-04') |  |
| Definitions | Paritition:  Portion of a large distributed dataset  How is # of partitions determined   * size of data * underlying partitioning of data * cluster configuration   Thread: Single process running on a core  Broadcast: Transfer the smaller of two tables to the larger, minimizing data transfer | |
| CACHE | CACHE TABLE fireCalls | Stores the table data in cache storage (can be checked under cluster/spark) |
| UNCACHE | UNCACHE TABLE fireCalls |  |
| CACHE LAZY | CACHE LAZY TABLE fireCalls |  |
| CLEAR CACHE | CLEAR CACHE |  |
| Set spark sql shuffle partition | SET spark.sql.shuffle.partitions=8 | Reduced runtime for same query  Less data with more stages may result in more compulational expense |
| Spark Jobs | Can check query runtime under jobs Spark UI |  |
| DESCRIBE | DESCRIBE fireCalls | Shows data type and comment |
| Adaptive Query Execution (AQE) | Uses Uses runtime statistics to detect skew from paritition sizes and split skew parititons into smaller subparititions | |
| SET AQE TRUE/FALSE | SET spark.sql.adaptive.enabled = TRUE |  |
| Two workloads | OLAP = Online Analytical Processing   * reporting * ad hoc analysis * blob stores * data warehouses   OLTP = Online Transaction Processing   * database * web traffic * streaming | |
| Define AWS Credentials | %python  ACCESS\_KEY = "AKIAJBRYNXGHORDHZB4A"  # Encode the Secret Key to remove any "/" characters  SECRET\_KEY = "a0BzE1bSegfydr3%2FGE3LSPM6uIV5A4hOUfpH8aFF".replace("/", "%2F")  AWS\_BUCKET\_NAME = "davis-dsv1071/data/"  MOUNT\_NAME = "/mnt/davis-tmp" |  |
| Mount the bucket | %python  try:    dbutils.fs.mount("s3a://{}:{}@{}".format(ACCESS\_KEY, SECRET\_KEY, AWS\_BUCKET\_NAME), MOUNT\_NAME)  except:    print("""{} already mounted. Unmount using `dbutils.fs.unmount("{}")` to unmount first""".format(MOUNT\_NAME, MOUNT\_NAME)) |  |
| Explore the mount | %fs ls /mnt/davis-tmp  %fs mounts |  |
| Unmount the data | %python  dbutils.fs.unmount("/mnt/davis-tmp") |  |
| driver | %scala  Class.forName("org.postgresql.Driver") |  |
|  | DROP TABLE IF EXISTS twitterJDBC;  CREATE TABLE IF NOT EXISTS twitterJDBC  USING org.apache.spark.sql.jdbc  OPTIONS (    driver "org.postgresql.Driver",    url "jdbc:postgresql://server1.databricks.training:5432/training",    user "readonly",    password "readonly",    dbtable "training.Account"  )  SELECT \* FROM twitterJDBC LIMIT 10 |  |
|  | DROP TABLE IF EXISTS twitterPhilippinesJDBC;  CREATE TABLE twitterPhilippinesJDBC  USING org.apache.spark.sql.jdbc  OPTIONS (    driver "org.postgresql.Driver",    url "jdbc:postgresql://server1.databricks.training:5432/training",    user "readonly",    password "readonly",    dbtable "(SELECT \* FROM training.Account WHERE location = 'Philippines') as subq"  )  %python  %timeit sql("SELECT \* from twitterJDBC").describe() | WHERE is the predicate pushdown  Sending filter deep down |
| CREATE TABLE TEMPORARY USING SEPERATOR : | CREATE OR REPLACE TEMPORARY VIEW fireCallsCSV  USING CSV  OPTIONS (      path "/mnt/davis/fire-calls/fire-calls-colon.txt",      header "true",      sep ":"    ) |  |
| Data types | bzip is splittable but gzip is not  Parquet is highly splittable and column based (so compressable) => faster |  |
| Delta | DROP TABLE IF EXISTS fireCallsDelta;  CREATE TABLE fireCallsDelta  USING delta  AS    SELECT /\*+ REPARTITION(8) \*/ \*    FROM fireCallsParquet |  |
| Read parquet file | spark.read.parquet() |  |
| Read CSV file | spark.read.csv() |  |
|  |  | |
| READ JSON | CREATE OR REPLACE TEMPORARY VIEW fireCallsJSON  USING JSON  OPTIONS (      path "/mnt/davis/fire-calls/fire-calls-truncated.json"    ) |  |
| CREATE TABLE | CREATE OR REPLACE TEMPORARY VIEW fireCallsJSON (    `Call Number` INT,    `Unit ID` STRING,    `Incident Number` INT,    `Call Type` STRING,    `Call Date` STRING,    `Watch Date` STRING,    `Received DtTm` STRING,    `Entry DtTm` STRING,    `Dispatch DtTm` STRING,    `Response DtTm` STRING,    `On Scene DtTm` STRING,    `Transport DtTm` STRING,    `Hospital DtTm` STRING,    `Call Final Disposition` STRING,    `Available DtTm` STRING,    `Address` STRING,    `City` STRING,    `Zipcode of Incident` INT,    `Battalion` STRING,    `Station Area` STRING,    `Box` STRING,    `Original Priority` STRING,    `Priority` STRING,    `Final Priority` INT,    `ALS Unit` BOOLEAN,    `Call Type Group` STRING,    `Number of Alarms` INT,    `Unit Type` STRING,    `Unit sequence in call dispatch` INT,    `Fire Prevention District` STRING,    `Supervisor District` STRING,    `Neighborhooods - Analysis Boundaries` STRING,    `Location` STRING,    `RowID` STRING  )  USING JSON  OPTIONS (      path "/mnt/davis/fire-calls/fire-calls-truncated.json"  ) |  |
| .write method | .write(" < path> ")  %python  df.write.mode("OVERWRITE").csv(username + "/fire-calls.csv") | Write using the .write(" < path> ") method on the DataFrame. |
| See the file | %python  dbutils.fs.ls(username + "/fire-calls.csv") |  |
| Check partitions | %python  df.rdd.getNumPartitions() |  |
|  | SELECT /\*+ COALESCE(n) \*/  CREATE OR REPLACE TEMPORARY VIEW fireCallsCSV1p    AS  SELECT /\*+ COALESCE(1) \*/ \*  FROM fireCallsCSV  SELECT /\*+ REPARTITION(n) \*/  CREATE OR REPLACE TEMPORARY VIEW fireCallsCSV8p    AS  SELECT /\*+ REPARTITION(8) \*/ \*  FROM fireCallsCSV | narrow transformation  reduce the #paritions - not even distribution  wide transformation  increase the #paritions - even distribution  we can change #partitions |
| Save results | %python  sql("SELECT \* FROM fireCallsCSV8p").write.mode("OVERWRITE").csv(username + "/fire-calls-repartitioned.csv") |  |
| View Results | %python  dbutils.fs.ls(username + "/fire-calls-repartitioned.csv") |  |
|  | A **table** creates a table in an existing database  A **view** is a SQL query stored in a database  Fastest is writing in table because query always does calculations  View can call query and calls the latest version of data always  A **global** **view** or table is available across all clusters  A **temporary** **view** or table is available only in the notebook you're working in | |
| TEMPORARY VIEW | DROP VIEW IF EXISTS groupedView;  CREATE OR REPLACE TEMPORARY VIEW groupedView    AS (      SELECT count(\*) AS count      FROM fireCallsParquet      GROUP BY Call\_Type\_Group      ORDER BY count    ) | Will be dropped when the session ends |
| TABLE | DROP TABLE IF EXISTS groupedTable;  CREATE TABLE IF NOT EXISTS groupedTable    AS (      SELECT count(\*) AS count      FROM fireCallsParquet      GROUP BY Call\_Type\_Group      ORDER BY count    ) |  |
|  | A **managed table** is a table that manages both the data itself as well as the metadata. In this case, a DROP TABLE command removes both the metadata for the table as well as the data itself.  **Unmanaged tables** manage the metadata from a table such as the schema and data location, but the data itself sits in a different location, often backed by a blob store like the Azure Blob or S3. Dropping an unmanaged table drops only the metadata associated with the table while the data itself remains in place. | |
| Managed Table | USE default;  DROP TABLE IF EXISTS tableManaged;  CREATE TABLE tableManaged (    var1 INT,    var2 INT  );  INSERT INTO tableManaged    VALUES (1, 1), (2, 2) |  |
| Unmanaged Table | DROP TABLE IF EXISTS tableUnmanaged;  CREATE EXTERNAL TABLE tableUnmanaged (    var1 INT,    var2 INT  )  STORED AS parquet  LOCATION '/tmp/unmanagedTable' |  |
| Check Floating Table | dbutils.fs.rm("/tmp/unmanagedTable", True) |  |
| Table View Tips | Use the appropriate view or table based on how you want to persist your query  Use external/unmanaged tables when you want to persist your data once the cluster has shut down  Use managed tables when you only want ephemeral data | |
| INSERT FROM | -- TODO  INSERT INTO newtable  SELECT `Address`, `City`, `Battalion`, `Box`  FROM fireCallsJSON  WHERE `Final Priority` = 3 |  |
| Check underlying details | %fs ls dbfs:/tmp/newTableLoc |  |
| CREATE NEW TABLE REPARITIONED FROM EXISTING TABLE | CREATE TABLE IF NOT EXISTS newTablePartitioned  AS  SELECT /\*+ REPARTITION(256) \*/ \*  FROM newTable |  |
| DROP TABLE | DROP TABLE newTable; |  |
|  | **Data Warehouses**  Pros: Great for BI  Cons: Limited Support for ML, Only with SQL interface  **Data Lakes**  Pros: Supports ML, Open formats and big ecosystems  Cons: Poor support for BI, Complex data quality problems | |
| Writing raw data into Delta Bronze | CREATE DATABASE IF NOT EXISTS Databricks;  USE Databricks;  DROP TABLE IF EXISTS fireCallsBronze;  CREATE TABLE fireCallsBronze  USING DELTA  AS    SELECT \* FROM fireCallsParquet |  |
| Refine bronze tables, write to Delta Silver | DROP TABLE IF EXISTS fireCallsSilver;  CREATE TABLE fireCallsSilver  USING DELTA  AS    SELECT Call\_Number, Call\_Type, Call\_Date, Received\_DtTm, Address, City, Zipcode\_of\_Incident, Unit\_Type, `Neighborhooods\_-\_Analysis\_Boundaries`    FROM fireCallsBronze    WHERE (City IS NOT null) AND (`Neighborhooods\_-\_Analysis\_Boundaries` <> "None");    SELECT \* FROM fireCallsSilver LIMIT 10; |  |
| UPDATE | UPDATE fireCallsSilver SET City = "San Francisco" WHERE (City = "SF") OR (City = "SAN FRANCISCO") |  |
| History | DESCRIBE HISTORY fireCallsSilver | Reflect History in Transaction Log |
| Aggregate data, write to Delta Gold | DROP TABLE IF EXISTS fireCallsGold;  CREATE TABLE fireCallsGold  USING DELTA  AS    SELECT `Neighborhooods\_-\_Analysis\_Boundaries` as Neighborhoods, Call\_Type, count(\*) as Count    FROM fireCallsSilver    GROUP BY Neighborhoods, Call\_Type |  |
| Paritioning by Column | CREATE DATABASE IF NOT EXISTS Databricks;  USE Databricks;  DROP TABLE IF EXISTS fireCallsDelta;  CREATE TABLE fireCallsDelta  USING DELTA  PARTITIONED BY (City)  AS    SELECT \* FROM fireCallsParquet | If the cardinality of a column will be very high, do not use that column for partitioning. For example, if you partition by a column userId and if there can be 1M distinct user IDs, then that is a bad partitioning strategy.  Amount of data in each partition: You can partition by a column if you expect data in that partition to be at least 1 GB. |
| INSERT OVERWRITE TABLE | INSERT OVERWRITE TABLE fireCallsDelta SELECT \*, `Neighborhooods\_-\_Analysis\_Boundaries` AS Neighborhoods FROM fireCallsDelta |  |
| AUTOMERGE | SET spark.databricks.delta.schema.autoMerge.enabled=TRUE |  |
| TIMETRAVEL | SELECT \* FROM table\_identifier TIMESTAMP AS OF timestamp\_expression  SELECT \* FROM table\_identifier VERSION AS OF version  SELECT \*  FROM fireCallsDelta    VERSION AS OF 0 |  |
| DELETE | DELETE FROM fireCallsDelta WHERE Incident\_Number = "14055109";  SELECT \* FROM fireCallsDelta WHERE Incident\_Number = "14055109" |  |
| Convert to Pandas DataFrame | %python  pdDF = sql("""SELECT timeDelay, Call\_Type, Fire\_Prevention\_District, `Neighborhooods\_-\_Analysis\_Boundaries`, Unit\_Type                FROM timeDelay                WHERE timeDelay < 15 AND timeDelay > 0""").toPandas() |  |
| Visualize | %python  import pandas as pd  import numpy as np  fig = pdDF.hist(column="timeDelay")[0][0]  display(fig.figure) |  |
| Train-Test Split | %python  from sklearn.model\_selection import train\_test\_split  X = pdDF.drop("timeDelay", axis=1)  y = pdDF["timeDelay"].values  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) |  |
| Baseline Model | %python  from sklearn.metrics import mean\_squared\_error  import numpy as np  avgDelay = np.full(y\_test.shape, np.mean(y\_train), dtype=float)  print("RMSE is {0}".format(np.sqrt(mean\_squared\_error(y\_test, avgDelay)))) |  |
| Build Linear Regression Model | %python  from sklearn.linear\_model import LinearRegression  from sklearn.preprocessing import OneHotEncoder  from sklearn.pipeline import Pipeline  ohe = ("ohe", OneHotEncoder(handle\_unknown="ignore"))  lr = ("lr", LinearRegression(fit\_intercept=True, normalize=True))  pipeline = Pipeline(steps = [ohe, lr]).fit(X\_train, y\_train)  y\_pred = pipeline.predict(X\_test) |  |
| See hot encoded feature names | %python  print(pipeline.steps[0][1].get\_feature\_names())  %python  coefs = pd.DataFrame([pipeline.steps[0][1].get\_feature\_names(), pipeline.steps[1][1].coef\_]).T  coefs.columns = ["features", "coef"]  coefs = coefs.sort\_values("coef", ascending=False)  coefs.head(10)  %python  coefs.sort\_values("coef", ascending=True).head(10) |  |
| Evaluate on Test Data | %python  from sklearn.metrics import mean\_squared\_error  print("RMSE is {0}".format(np.sqrt(mean\_squared\_error(y\_test, y\_pred)))) |  |
| Save Model | %python  try:    import mlflow    import mlflow.sklearn    with mlflow.start\_run(run\_name="Basic Linear Regression Experiment") as run:      mlflow.sklearn.log\_model(pipeline, "Model Pipeline")  except:    print("ERROR: This cell did not run, likely because you're not running the correct version of software. Please use a cluster with a runtime with `ML` at the end of the version name.") |  |